В осеннем семестре второго курса в рамках изучения языка программирования **C** будет выдано два практических задания. Первое задание является обязательным к выполнению для получения зачёта.

Выполнение задания практикума, как правило, включает в себя несколько этапов, которые нужно последовательно сдавать преподавателю. По каждому из этапов фиксируется крайний срок его сдачи. При этом не запрещается, а даже приветствуется сдача нескольких (хоть всех) этапов раньше срока. Однако, сдача всех этапов в крайний день сдачи последнего этапа формально не является допустимой. Таким образом, каждый из этапов должен быть сдан не позднее крайнего срока обозначенного преподавателем для этого этапа.

Задание 1 (интерпретатор команд)

Введение

В качестве первого задания практикума предлагается написать программу на языке **C** с использованием программного интерфейса (*API*) ОС семейства *Unix*. Данная программа представляет собой упрощённый интерпретатор пользовательских команд, похожий на *bash*. Отметим, что все стандартные команды, в числе которых *ls*, *pwd*, *chmod*, *rm*, *mv*, *cp*, *find*, *grep* и др. уже есть в системе и расположены в каталоге `/bin/`, т.е. реализовывать их не нужно, они будут вызываться при помощи функций ядра (*API*) ОС семейства *Unix*.

Описание интерпретатора команд

Логически выполнение программы интерпретатора команд включает в себя три строго последовательных шага, повторяемых в цикле от начала и до завершения программы.

На первом шаге в стандартный поток вывода программы (*STDOUT*) печатается приглашение к вводу команды, которое, обычно, оканчивается символом доллара `$`, а также содержит различную дополнительную информацию (имя или путь к текущей директории, имя пользователя и т.п.). Примерами приглашений к вводу являются: `scripts $`, `antontodua@antontodua:~/backend-cpp$` и др.

На следующем (втором) шаге интерпретатор целиком считывает команду пользователя со стандартного потока ввода (*STDIN*). Чтение команды обычно завершается символом перевода строки `\n`, который возникает при нажатии клавиши *ENTER*. Примерами пользовательских команд являются: `ls\n`, `cd ..\n`, `mv a.txt task.txt\n` и др.

Если команда не умещается в одну строчку допустимо использование символа обратного слэша `\` непосредственно перед переводом строки `\n`, который в этом случае будет проигнорирован. Другими словами, команды `mv a.txt task.txt\n`, `mv a.txt\\n task.txt\n` и `mv a.txt \\ntask.txt\n` являются эквивалентными.

Текст команды состоит из нескольких слов (***токенов***), разделённых одним или несколькими пробельными символами. Первое слово задаёт исполняемый файл (программу), которая будет запущена на шаге выполнения команды. Второе и последующие слова задают аргументы командной строки, которые будут переданы программе при запуске. Разумеется, передача аргументов командной строки не является обязательной. К примеру, команда `ls\n` не содержит аргументов командной строки, а команда `mv a.txt task.txt\n` — содержит сразу два — `a.txt` и `task.txt`.

Иногда бывает так, что в аргументе команды хочется передать пробельный символ или же один из специальных символов (вроде вертикальной черты `|`). В подобных случаях аргумент команды можно частично или полностью заключить в кавычки (`"` двойные или `'` одинарные). К примеру, следующие вызова команды `echo` с одним аргументом полностью эквивалентны: `echo "Hello, World!"`, `echo 'Hello, World!'`, `echo Hello,' 'World!`, `echo Hello', 'World!`, `echo Hello", "World!`, `echo "Hello, "World!`, `echo Hello', World!'`. Отличие двойных `"` кавычек от одинарных `'` состоит в том, что двойные кавычки допускают экранирование символов внутри (по аналогии со строковыми константами в языке **C**), а именно экранирование самих двойных кавычек `\"` и символа обратного слэша `\\`.

На последнем (третьем) шаге интерпретатор выполняет команду, введённую пользователем на предыдущем шаге, или сообщает пользователю о том, что выполнение невозможно, т.к. команда содержит ошибки. Наиболее распространённой является ошибка `-bash: abrakadabra: command not found`, означающая, что команда с именем `abrakadabra` не найдена. Непосредственно выполнение команды представляет собой запуск соответствующего исполняемого файла, расположенного в одном из стандартных каталогов (например, `/bin/`) с заданными аргументами командной строки (считанными на втором шаге).

Требование к выполнению задания

Разработку интерпретатора команд логически можно разделить на пять этапов:

1. Печать стандартного приглашения к вводу команды и чтение команды (перевод текста команды в некоторую программную структуру данных, пригодную для дальнейшего вызова соответствующего исполняемого файла)
2. Выполнение команды, прочитанной на предыдущем шаге, включая команды `cd` — смена текущей директории и `exit` — завершение работы интерпретатора
3. Поддержка режима выполнения команд в фоновом режиме с помощью символа амперсанда `&`, например: `echo a & echo b & echo c`
4. Поддержка перенаправления стандартных потоков ввода и вывода с помощью символов `<`, `>` и `>>`, например: `echo abc >> abc.txt`
5. Поддержка конвейерного режима выполнения команд с помощью символа вертикальной черты `|`, например: `find -type f -name '*.swp' | xargs rm -rf`

Стоит отметить, что для сдачи первого этапа задания достаточно считать текст команды в структуру достаточную для реализации второго этапа, т.е. обрабатывать символы амперсанда `&`, перенаправления стандартных потоков `<`, `>`, `>>` и вертикальной черты `|` не требуется. Однако, выполнение первого этапа с учётом данных особенностей, позволит сильно облегчить разработку последующих этапов.

Ещё раз подчеркнём, что сами команды (вроде `cat`) не нужно реализовывать, они уже реализованы! Для первого этапа достаточно просто считать со стандартного потока ввода массив слов одной команды. К примеру, если пользователь ввёл команду `find -type f -name '*.swp'`, то результат чтения этой команды можно сохранить в виде массива из 5 указателей на соответствующие строки (аргументы) команды: `find`, `-type`, `f`, `-name`, `*.swp`.

Для выполнения второго этапа задания также потребуются некоторые функции ядра (*API*) ОС семейства *Unix*. Объявления необходимых функций располагаются в заголовочном файле `#include <unistd.h>`. В число функций, с помощью которых будет выполняться задание входят: `fork`, `execvp`, `wait` и др.